Class 16: Recursive Data Types

Schedule

You should read MCS Chapter 7 this week.

Problem Set 7 is due **28 October (Friday) at 6:29pm**.

Structural Induction

We can show a property holds for *all objects of a data type* by:

1. Showing the property holds for all base objects.
2. Showing that all the ways of constructing new objects, preserve the property.

What is the difference between scalar data and compound data structures?

Pairs

Definition. A *Pair* is a datatype that supports these three operations:

$$\begin{aligned}
&\textit{make_pair} : \textit{Object} \times \textit{Object} \rightarrow \textit{Pair} \\
&\textit{pair_first} : \textit{Pair} \rightarrow \textit{Object} \\
&\textit{pair_last} : \textit{Pair} \rightarrow \textit{Object}
\end{aligned}$$

where, for any objects a and b, $\textit{pair\_first}(\textit{make\_pair}(a, b)) = a$ and $\textit{pair\_last}(\textit{make\_pair}(a, b)) = b$.

Lists

Definition (1). A *List* is either (1) a *Pair* where the second part of the pair is a *List*, or (2) the empty list.

Definition (2). A *List* is a ordered sequence of objects.

List Operations

$$\begin{aligned}
\mathit{first}(\mathit{prepend}(e, l)) &= \underline{\hspace{2em}} \\
\mathit{rest}(\mathit{prepend}(e, l)) &= \underline{\hspace{2em}} \\
\mathit{empty}(\mathit{prepend}(e, l)) &= \textbf{False} \\
\mathit{empty}(\underline{\hspace{2em}}) &= \textbf{True}
\end{aligned}$$

Definition. The *length* of a list, p, is:

$$\begin{cases} 0 & p \text{ is } \textbf{null} \\ \underline{\hspace{12em}} & p = \mathit{prepend}(e, q) \text{ for some object } e \text{ and some list } q \end{cases}$$

Unique Construction property:

$$\forall p \in \mathit{List} - \{\textbf{null}\}.\ \exists q \in \mathit{List}, e \in \mathit{Object}.\ p = \mathit{prepend}(e, q) \land$$
$$(\forall r \in \mathit{List}, f \in \mathit{Object}.\ p = \mathit{prepend}(f, r) \implies r = q \land f = e)$$

Why is this necessary for our length definition?

```
def list_prepend(e, l):
    return make_pair(e, l)

def list_first(l):
    return pair_first(l)

def list_rest(l):
    return pair_last(l)

def list_empty(l):
    return l == None

def list_length(l):
    if list_empty(l):
        return 0
    else:
        return 1 + list_length(list_rest(l))
```

Prove: for all lists, p, `list_length(p)` returns the length of the list p.

